We described the simple programming language **While** that is to be used as a basis for the discussion in the rest of the module.
The meaning of an expression depends on the values bound to the variables that occur in it.

**Example**

If \( x \) is bound to 3 then the arithmetic expression \( x + 1 \) evaluates to 4.
We need to introduce the concept of *state*: to each variable the state associates its current value.

We shall represent a state as a function from variables to values, *i.e.* it is an element of the set

\[
\text{State} = \text{Var} \rightarrow \mathbb{Z}
\]

Each state specifies a value, written \( s \, x \), for each variable \( x \) of \( \text{Var} \).

Therefore, if \( s \, x = 3 \) then the value of \( x + 1 \) in the state \( s \) is 4.
There are several alternative ways to think about the state. One is as a table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

or as a “list” of the form

\[ [x \mapsto 5, \ y \mapsto 7, \ z \mapsto 0] \]

In all cases we must ensure that there is only one value associated with each variable. By requiring the state to be a function this requirement is trivially fulfilled, whereas for the alternatives extra conditions have to be enforced.
Given an arithmetic expression $a$ and a state $s$ we can determine the value of the expression.

The meaning of an arithmetic expression is given by the semantic function:

$$ \mathcal{A} : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) $$

This means that $\mathcal{A}$ takes its arguments one at a time. We can therefore supply $\mathcal{A}$ with its first parameter, say $x + 1$, and study the function $\mathcal{A}[x + 1]$. It has functionality $\text{State} \rightarrow \mathbb{Z}$ and only when we supply it with a state do we obtain the value of the expression $x + 1$. 
The definition of $A$ is given in Table 1.

\[
\begin{align*}
A[n] s &= N[n] \\
A[x] s &= s \times \\
A[a_1 + a_2] s &= A[a_1] s + A[a_2] s \\
A[a_1 \times a_2] s &= A[a_1] s \times A[a_2] s \\
A[a_1 - a_2] s &= A[a_1] s - A[a_2] s
\end{align*}
\]

Table: The Semantics of Arithmetic Expressions
Semantic Functions

Semantic Function $\mathcal{B}$

We can define

$$\mathcal{B} : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathcal{T})$$

[where $\mathcal{T}$ consists of the two truth values $\text{tt}$ (for true) and $\text{ff}$ (for false)] by
the semantic clauses of Table 2.

$$\begin{align*}
\mathcal{B}[\text{true}] s &= \text{tt} \\
\mathcal{B}[\text{false}] s &= \text{ff} \\
\mathcal{B}[a_1 = a_2] s &= \begin{cases} 
\text{tt} & \text{if } A[a_1] s = A[a_2] s \\
\text{ff} & \text{if } A[a_1] s \neq A[a_2] s \end{cases} \\
\mathcal{B}[a_1 \leq a_2] s &= \begin{cases} 
\text{tt} & \text{if } A[a_1] s \leq A[a_2] s \\
\text{ff} & \text{if } A[a_1] s > A[a_2] s \end{cases} \\
\mathcal{B}[- b] s &= \begin{cases} 
\text{tt} & \text{if } \mathcal{B}[b] s = \text{ff} \\
\text{ff} & \text{if } \mathcal{B}[b] s = \text{tt} \end{cases} \\
\mathcal{B}[b_1 \land b_2] s &= \begin{cases} 
\text{tt} & \text{if } \mathcal{B}[b_1] s \text{ and } \mathcal{B}[b_2] s \\
\text{ff} & \text{if } \text{not} (\mathcal{B}[b_1] s \text{ and } \mathcal{B}[b_2] s) \end{cases}
\end{align*}$$
The lecture was based on Sections 1.3 and 1.4 of the module textbook [NN92].
In this lecture we describe the meaning of arithmetic and boolean expressions.
Natural semantics are concerned with the relationship between the *initial* state and the *final* state of an execution. The lecture will be based on Section 2.1 of the module textbook [NN92].
Bibliography

H.R. Nielson and F. Nielson.
Semantics with Applications: A Formal Introduction.